Bayesian Network Review

Linjian Li

1. Introduction

In the machine learning area, classification is a basic task which needs the system to predict a label for a data instance given some attributes (or features). Plenty of models and algorithms have been proposed to accomplish this task, such as support vector machine (SVM), decision tree, neural network, Bayesian methods and so on.

Naïve Bayes classifier is a model which makes use of the Bayes' theorem to do classification. The Bayes' theorem can be simply described by the following equation:

$$\mathrm{P(A|B)} = \frac{\mathrm{P(B|A)P(A)}}{\mathrm{P(B)}}$$

where A and B are events and $P(B) \neq 0$. P(A|B) is the conditional probability meaning that the likelihood of event A occurring given that B is true. P(B|A) is similar to P(A|B). P(A) and P(B) are the probabilities of observing A and B independently of each other. Naïve Bayes classifier has been used to solve many practical problems like text classification and spam detection and it has shown the strong power of itself. Although the naïve Bayes classifier has shown the power in solving practical problems, it makes a strong independence assumption which is impractical. Strong independence assumption can be described as:

$$P(A_i, A_j|C) = P(A_i|C)P(A_j|C)$$

It means all the attributes A_i are conditionally independent given the class label C. This assumption is barely in the real world and therefore the naïve Bayes classifier is naïve.

The naïve strong independence assumption may stop the model to achieve better performance. So, a new model, Bayesian network, was proposed by Pearl in 1988, trying to improve the performance by dropping the string independence assumption. Bayesian networks are directed acyclic graphs in which the nodes and arcs represent variables and direct causal influences between linked variables. In this review, we will use the words "node" and "variable" equivalently. The strengths of the influences between variables are expressed by the conditional probability distribution (CPD) of each variable. The conditional probability distribution of each variable describes the probabilities of possible values of this variable given its parents. If the variable is discrete, this can be expressed in the form of conditional probability table (CPT). In the rest of this paper, the Bayesian network we discuss will only contain discrete variable nodes, unless otherwise stated.

Bayesian network drops the strong independence assumption of variables but keeps a weak independence assumption, which is: each variable x_i in Bayesian network is conditionally independent of all its non-descendants, given its all parents Π_i. Combining the weak independence assumption and the chain rule, the joint probability distribution, also called global distribution, of the whole Bayesian network is determined by the CPD, also called local distribution, of each variable. Therefore we can use much fewer parameters to describe the network with these CPDs than describing it directly with a joint probability distribution. The joint probability distribution can be calculated by CPDs as

$$P(X) = \prod_{i=1}^{n} P(x_i|\Pi_i)$$

where n is the number of variables and X is the set of all variables $\{x_1, x_2, \dots, x_n\}$. Another advantage of decomposing the joint probability distribution into local CPDs is that it makes local computation possible for tasks like inference. Inferencing by directly computing the joint probability is NP-hard [10] [15].

Having the definition of Bayesian network, we can find out that the naïve Bayes classifier is a special case

of Bayesian network, in which all the arcs are from the class variable to the attribute variables, and there is no arc between any two attribute variables. If the variables are continuous, users need to discretize them with Weka.

2. Related Works

Scutari developed an R package called "bnlearn" [30]. The bnlearn package implements several constraint-based structure learning algorithms and a score-based learning algorithm, with the ability to learn from discrete and continuous data.

Dlib is a C++ library which includes tools for Bayesian network [31]. But the functionality of this library is limited. We can only use this library to do simple operations.

An open-source Python package called "bayesian-belief-netowrks" is developed by eBay [32], supporting for discrete and continuous variables. But the package needs the users to specify the Bayesian network structure instead of learning the structure automatically.

The well-known open-source machine learning software called "Weka" has implement various Bayesian network classifier learning algorithms [33]. A great advantage of it is that it provides GUI which makes inspection much easier. But Weka only support Bayesian networks with discrete variables.

An open-source software for learning Bayesian networks called "URLearning" [34], developed by Yuan et al., formulates the structure learning problem as a shortest path problem and uses various heuristic search algorithms to learn the structure. The repository also provides datasets for many common Bayesian network structures [35].

3. Bayesian Network Structure Learning

Given a bunch of nodes representing attribute variables, in order to use Bayesian network to model a real-world problem, we need to add arcs between these nodes. But how can we know which arcs should be added and which arcs should not? In general, there are two approaches. The first one is to specify the Bayesian network structure by expert knowledge, which can be quite subjective. Thus, we will not discuss this approach in this review. The second one is to learn the structure from the training data and the task is called Bayesian network structure learning (BNSL).

There are two ways to learn the structure from data: constraint-based and score-based. Constraint-based methods work by checking whether the conditional independences among the variables in the training data hold in the Bayesian network. Score-based methods work by using the outputs of some metrics, called scoring functions, to determine the fitness of the Bayesian network on the data.

2.1. Scoring Functions

Given n nodes, there are 2^n possible underlying undirected graph structure and thus the number of possible directed acyclic graphs is $\Theta(2^n)$. If we assume that we can specify an ordering of all n nodes, such that, for any $i < j$, we forbid the arc from x_j to x_i, i.e., the parent(s) of a node must precede this node in the ordering we specified. Subject to this ordering, there are $2^{\binom{n}{2}} = 2^{n(n-1)/2}$ possible Bayesian network structures [2].

To find the best or the top-k best structure(s) of the Bayesian network, we need some metrics that enable us to compare the qualities of different structures.

To describe the metrics, we introduce some notations. Given a Bayesian network structure B_S, let n be the number of the variables in B_S, $X = \{x_1, x_2, \dots, x_n\}$ be the set of these variables, x_i ($1 \le i \le n$) be one of the variables, and Π_i be the set of parents of x_i. Different variables may have different number of possible values (or states), so we let r_i be the number of values of variable x_i, and $q_i = \prod_{x_l \in \Pi_i} r_l$ be the number of values of Π_i. Let θ_{ijk} represent the $P(x_i = k | \Pi_i = j)$ which means the probability that $x_i = k$ given the j-th value of the

x_i's parents. Let N_{ijk} be the number of instances in the training set such that $x_i = k$ and x_i's parents' value is the j-th value. Let $\mathcal{D}$ be the training dataset.

A class of commonly used metrics is known as scoring functions, whose inputs are a Bayesian network structure and the training dataset, measuring the ability of the Bayesian network to represent the distribution of the data. Another explanation of scoring function is: given a training dataset in which the instances are drawn from an unknown underlying distribution, the scoring function calculates the probability that the underlying distribution can be expressed by the Bayesian network structure, i.e.

$$\mathrm{Score}(B_S, \mathcal{D}) = P(B_S|\mathcal{D})$$

and the goal is to maximize the score. Given a fixed dataset $\mathcal{D}$, $P(\mathcal{D})$ is a constant. So,

$$\mathrm{Score}(B_S, \mathcal{D}) \propto P(B_S|\mathcal{D}) \cdot P(\mathcal{D}) = P(B_S, D).$$

Thus, maximize the score is equivalent to maximize $P(B_S, D)$.

One important feature of scoring functions is the decomposability, which is that the score of the whole Bayesian network can be computed by adding the local scores of disjoint sub-networks, i.e.,

$$\mathrm{Score}(B_S, D) = \sum_{i=1}^{n} Score(x_i, \Pi_i|D)$$

Below, we will present some commonly used scoring functions.

- Bayesian Dirichlet (BD)

  This is proposed by Heckerman et al. in [3]. The equation can be written as

  $$p(D, B_s^h|\xi) = p(B_s^h|\xi) \cdot \prod_{i=1}^{n}\prod_{j=1}^{q_i} \frac{\Gamma(N'_{ij})}{\Gamma(N'_{ij}+N_{ij})} \prod_{k=1}^{r_i} \frac{\Gamma(N'_{ijk}+N_{ijk})}{\Gamma(N'_{ijk})}$$

  (h 和 ξ是在原文里面出现的，我现在只是截个图，其实在这里可以去掉) where $P(B_S)$ is the prior of B_S, which usually assumed to be equal, for simplicity, for all possible B_S, Γ is the Gamma function, N'_{ijk} a hyperparameter of Dirichlet distribution. Unfortunately, specifying N'_{ijk} for all possible variable-parent configurations and for all i, j and k conditionally impossible. We need some method to assign them values.

- K2

  Proposed by Cooper and Herskovits in [2]. The equation of $P(B_S, D)$ is

  $$P(B_S, D) = P(B_S) \prod_{i=1}^{n}\prod_{j=1}^{q_i} \frac{(r_i-1)!}{(N_{ij}+r_i-1)!} \prod_{k=1}^{r_i} N_{ijk}!$$

  It is a more special case of BD. If we uniformly set $N'_{ijk} = 1$ in the equation of BD, then we get the equation of K2.

- BDe

  BDe is also proposed in [3] and is a likelihood-equivalent specialization of BD metric by setting

  $$N'_{ijk} = N' \cdot P(x_i = k, \Pi_i = j|B_S)$$

  where N' is equivalent sample size. The "e" in "BDe" is for likelihood-equivalence.

- BDeu

  In BDe, we need to know $P(x_i = k, \Pi_i = j|B_S)$ for all i, j, and k, which might be hard to know. A simple way to calculate the value is

  $$P(x_i = k, \Pi_i = j|B_S) = \frac{1}{r_i q_i}$$

  This is called BDeu and was originally proposed by Buntine in [4]. The "u" in "BDeu" is for uniform joint distribution, because every state of the joint space (i.e. every configuration of j and k), conditioned on B_S, is equally likely.

- Log-Likelihood

  Proposed in [5]. The equation is

  $$\mathrm{LL}(B_S|D) = \sum_{i=1}^{n}\sum_{j=1}^{q_i}\sum_{k=1}^{r_i} N_{ijk} \cdot \log\left(\frac{N_{ijk}}{N_{ij}}\right).$$

  However, the simple log-likelihood score tends to favour those structures with more arcs rather than fewer arcs.

- Minimum Description Length (MDL)

  The earliest idea was proposed in [6], but was discussed clearer in [7]. The equation can be written as

  $$\mathrm{MDL}(B_S|D) = \mathrm{LL}(B_S|D) - \frac{1}{2}\log(N)|B_S|$$

  where $|B_S| = \sum_{i=1}^{n}(r_i - 1)q_i$ is the network complexity, which is the number of parameters for network B_S.

- Bayesian Information Criterion (BIC)

  The equation is the same as MDL.

- Akaike Information Criterion (AIC)

  Proposed in [8] and the equation is

$$\mathrm{AIC}(\mathrm{B_S}|D) = \mathrm{LL}(\mathrm{B_S}|D) - |B_S|$$

which is similar to MDL.

- Normalized Minimum Likelihood

  The scoring functions based on information theory are essentially in the form of

  $$\mathrm{Score}(\mathrm{B_S}|D) = -\big(L(D|B) + L(B_S)\big)$$

  where $L(D|B)$ represents the length of bits to describing dataset *D* with the Bayesian network $\mathrm{B_S}$ and $\mathrm{L}(\mathrm{B_S})$ represents the length to describe the Bayesian network $\mathrm{B_S}$. As shown above, MDL, BIC, and AIC all choose to set $L(D|B_S) = -LL(B_S|D)$. But their choices of $\mathrm{L}(\mathrm{B_S})$ are different. MDL and BIC choose $\frac{1}{2}\log(N)|B_S|$ while AIC chooses $|B_S|$. It has never been consensual that which setting of $\mathrm{L}(\mathrm{B_S})$ is better. The NML tries to give a better solution by minimizing the worst-case regret (in decision theory). We will not discuss it in detail. Interested readers may refer to [9] for more details.

2.2. Structure Learning Algorithms

Given a set of nodes, a trivial idea to find the best structure is to compute the scores for all possible structures and then pick up the top one. However, as stated at the beginning of section 2.1., the number of possible structures is at least $2^{\binom{n}{2}} = 2^{n(n-1)/2}$, where *n* is the number of nodes. It is impractical to apply this trivial idea in practice. We need some more efficient methods to learn the structure.

Chow and Liu proposed an algorithm [11], we call it Chow-Liu algorithm, that can learn a Bayesian network in the form of a tree in polynomial time. Chow-Liu algorithm computes mutual information for each pair of nodes forming a complete graph. Then generate a maximum spanning tree with the mutual information as the weights of the arcs. The tree is undirected, so we choose a node as the root node and add direction for each arc away from the root. The main cost for Chow-Liu algorithm is spent on computing the mutual information for each pair of nodes. There are *n(n-1)/2* pairs given *n* nodes and computing mutual information for one pair needs to iterate the whole dataset with *N* data instances. Thus, the time complexity for Chow-Liu algorithm is $\mathrm{O}(\mathrm{n}^2 N)$.

The weakness of Chow-Liu algorithm is that it restricts the shape of the Bayesian network to be a tree. Silander and Myllymaki proposed an algorithm [14] that learn the structure by first calculating the local scores for all possible variable sets and find the best parents set for each variable. Then find an ordering of the variable sets. Using the found ordering and the best parents set for each variable to find a best network structure. However, because the problem is essentially NP-hard, the computational complexity of this algorithm is $\mathrm{O}(\mathrm{n}^2 2^{n-2})$, which is still in exponential time. And due to the high computational complexity, the algorithm grantee feasibility only for $n < 33$.

To learn a general shape structure with more variables, many works have adopted the methods based on the idea of dynamic programming. Some works learn the global structure by first learn the local structures for the nodes and then combine the local structures into a global structure.

Niinimaki and Parviainen proposed a score-based local-to-global structure learning algorithm [12]. The algorithm chooses some target variables and then learn the Markov blankets of these variables. These Markov blankets are local structures. And combine all these local structures, with AND-rule to resolve the conflicts, to form a global structure.

However, using AND-rule to resolve conflicts may prune some parents or children that should have present. To address the problem above, Gao et al. proposed a new local-to-global method [22][13] by starting at a random node and gradually expands the learned structure until the global structure is learned. Their algorithm does not need to revisit lots of learned nodes and drops the need of AND-rule, therefore is more efficient.

Based on their own previous works, Gao et al. proposed an improved algorithm [22]. The new algorithm interacts with the learning history making it computational saving and possible to be implemented in a parallel manner.

Many works also adopted the idea that learning a structure with the constraint of a topological ordering, such that the parents of a node will not appear after this node in the ordering.

Ott et al. proposed an algorithm [17] that transform the problem of finding a best network into the problem of finding a best ordering (the original word in that paper is "permutation"). The algorithm measures the quality of the ordering by the best score of the network whose topological ordering is consistent with that ordering. After finding the optimal ordering, the algorithm can then find an optimal structure with the ordering.

Tamada et al. proposed an algorithm [19] which realizes direct parallel of the previous dynamic programming algorithm, proposed by Ott et al., with tolerable time and space overheads. The contribution of their work is that they proposed the algorithm to number the processors used in parallel computing and to reduce the overhead of communication between processors. Thus, their algorithm can run efficiently in parallel.

Koivisto and Sood proposed an algorithm [18] that calculates the posterior probability of a network by summing over the ordering with which the network is consistent. Then select the argmax ordering and network.

Ellis and Wong proposed an MCMC-based (Markov chain Monte Carlo) algorithm [20] to reduce the complexity of structure learning. The algorithm can start at a random graph structure and use MCMC method to move to a new state of structure with a certain probability. However, when the number of nodes is large, this method will easily be trapped in local optima. Thus, Ellis and Wong proposed to use MCMC to sample the ordering instead of the graph structure directly, to escape the local optima. Then samples a graph structure from the selected ordering.

Wang et al. proposed two algorithmic improvements [21] to the MCMC-based structure learning algorithm. The first improvement is that the proposed algorithms that map between k-combinations and integers, which make parallel learning by assigning tasks to different GPU threads easier. The new mapping algorithms are much simpler than traditional hashing methods. The second improvement is the new scoring function they proposed for the orderings. The new max-based scoring function simplifies the computation. The new scoring function drops the requirement of enumerating all possible graph structure and thus significantly saves the computational cost.

There are maybe many Bayesian networks having the scores very close to optimal, thus, instead of finding only one "optimal" Bayesian network, many works try to find the top-k network and take the average to improve the performance. Liao et al. proposed an algorithm to find all the network structures within a factor of optimal [28], instead of restricting the number k. Their algorithm scales to the network with almost 60 variables while previous works usually scale to network with fewer than 30 variables.

The disadvantage of most algorithms learning top-k Bayesian networks is that the top-k networks are usually very similar to each other. They tend to be trapped in the same local optima. To address this problem, Chen and Yuan proposed an algorithm to learn diverse Bayesian networks [29], which is to find several networks that are significantly different from each other. To find diverse Bayesian networks, the algorithm first find the diverse The found networks are from different local optima, so the algorithm is more likely to reach the global optima.

Apart from using score-based methods only, Tsamardinos et al. proposed a hybrid algorithm named Max-Min Hill-Climbing (MMHC) [16] using both constraint-based methods and score-based methods. MMHC first learns the skeleton, which is an undirected graph, of a Bayesian network using a local discovery algorithm named Max-Min Parents and Children (MMPC). MMHC calls MMPC on each variable to find the correspond parents-children sets for all variables and then form the skeleton. MMHC then orients the skeleton using a greedy Bayesian-scoring hill-climbing search. MMHC trades-off time for high memory and high-dimensional dataset requirements. An advantage of MMHC is that it uses

score-based method to orient the skeleton, dropping the requirement that previous works need, which is that the user must specify the maximum number of parents for each variable. Another advantage of MMHC is that it can easily be modified to selectively reconstruct only a part of the network.

Campos and Ji [36] proposed to reduce the search space with some structural constraint and to strongly reduce the time and memory costs using properties of decomposable scoring functions. They also proposed a branch-and-bound approach with caches to produce an anytime solution.

4. Learning Parameters

Only having the structure of a Bayesian network is not enough. We also need the parameters, which are the CPDs in the network, to make the inferences. To learn the parameters from the training dataset, the simplest and the most commonly used method is to simply calculate the frequency and divide it by the frequency of instances that are compatible on the parent variables. The equation can be expressed as

$$\mathrm{P}(\mathrm{x_i} = \mathrm{k}|\Pi_\mathrm{i} = \mathrm{j}) = \frac{N_{ijk}}{N_{ij}}$$

Usually, we will adopt some smoothing techniques to avoid the parameters' value being equal to zero.

5. Inference

Having learned the structure and the parameters of a Bayesian network, we can use this model to make inferences given some evidence (observations on some variables). Essentially, making inferences means to calculate the posterior probability distribution $P(Y|X)$ where Y are the variables that we are interested in and X are the variables that we have observed. There are two class of inference methods, one is exact inference and the other one is approximate inference.

4.1. Exact Inference

As stated earlier, the joint probability distribution is decomposed to the local CPDs in the Bayesian network, and we can restore the joint probability distribution by multiplying all CPDs together. Having the joint probability distribution, we can then sum out all the variables except the target variables and the observed variables. After normalization, we can get the posterior probability of the target variables and make the inference. However, when the number of variables and the number of possible values of each variable are large, the space of the joint probability will be huge, making the computational cost, in space and time, unaffordable. In fact, the general problem of inference in Bayesian network has been proved to be NP-hard by Cooper [23]. Besides, if we want to use the joint probability distribution directly, there is no need to use the model of Bayesian network. Thus, we need some more efficient methods to make inferences making full use of the CPDs in the Bayesian network.

Lauritzen and Spiegelhalter proposed a method [24] which we call the junction tree (some may call it clique tree) algorithm. The junction tree algorithm combines multiple nodes in the Bayesian network into one "clique". By constructing several cliques and connecting them together, using two methods called *moralization* and *triangulation*, a junction tree (clique tree) is formed. Having the junction tree, and given the evidence of some variables, the algorithm uses a method called belief propagation to pass the messages about the evidence through the whole junction tree. Belief propagation consists of two sub-routine which are *collect* and *distribute*. After belief propagation, each clique in the junction tree will have the correct conditional probability distribution of the variables in that clique, conditional on the evidence. The advantage of junction tree algorithm is that after running the algorithm given some evidence, we will have the conditional probability distribution of all the other variables conditional on the evidence. So, we can make inferences on many variables without re-run the algorithm if there is no more new evidence. However, this feature of junction tree algorithm also

causes the disadvantage of it. Facing the practical problem, we usually do not care about every variable but only some few variables. Junction tree algorithm usually spent lots of computational resources and time on the variables that we do not care about.

Zhang and Poole proposed an algorithm called variable elimination algorithm [25] to overcome the disadvantage of junction tree algorithm in terms of pruning irrelevant variables. One of the contributions of this work is that it defines the standards of irrelevant variables. The algorithm first prunes the irrelevant variables, which significantly reduce the computational cost. Then, given an elimination ordering, the algorithm eliminates the variables by summing them out one by one in the corresponding CPDs. The disadvantage of variable elimination algorithm is that it needs a good elimination ordering to run efficiently. And given different target variables and evidence variables, the optimal choice of elimination ordering may be different, so we need to determine the elimination ordering each time we want to make inference.

4.2. Approximate Inference

The most common way to make approximate inferences is to sample the Bayesian network.

Henrion proposed probabilistic logic sampling algorithm [26]. The algorithm draws a number of samples from the network and drops the samples which conflict with the evidence. Then, the distribution of target variables in the remaining samples is the distribution conditional on the evidence.

Another method is MCMC with Gibbs sampling [27]. After drawing the initial sample, the algorithm selects a new variable at random (while it usually selects in the order in implementation), and draws a new sample from conditional probability distribution while fixing all the other variables and changing only the selected variable. The initial several samples are usually abandoned in a period called “burn-in”, and keep all the samples after the burn-in period.

Likelihood weighting [37, 38] is also a very commonly used method to make approximate inferences in Bayesian network. Given evidence, the algorithm of drawing one sample and the corresponding weight in the following way:

1. initialize weight $w = 1$
2. for each node (variable) X in the **topological ordering** of the Bayesian network
   2.1. if the node is in the evidence
      2.1.1. set the value of this variable to be consistent with the evidence
      2.1.2. $w = w * P(X=\text{value}|\text{parents}(X))$
   2.2. else
      2.2.1. draw a random sample from $P(X|\text{parents}(X))$ for variable X

The algorithm draws samples of variables in the topological ordering of the Bayesian network, which makes sure that when drawing sample for any variable, the values of its parents (if it has parents) have been drawn and can be used in computation of conditional probabilities.

6. References

[1] Pearl, J. (2014). Probabilistic reasoning in intelligent systems: networks of plausible inference. Elsevier.

[2] Cooper, G. F., & Herskovits, E. (1992). A Bayesian method for the induction of probabilistic networks from data. *Machine learning*, *9*(4), 309-347.

[3] Heckerman, D., Geiger, D., & Chickering, D. M. (1995). Learning Bayesian networks: The combination of knowledge and statistical data. *Machine learning*, *20*(3), 197-243.

[4] Buntine, W. (1991, July). Theory refinement on Bayesian networks. In *Proceedings of the Seventh conference on Uncertainty in Artificial Intelligence* (pp. 52-60). Morgan Kaufmann Publishers Inc..

[5] Bouckaert, R. R. (1995). *Bayesian belief networks: from construction to inference* (Doctoral dissertation).

[6] Suzuki, J. (1993, January). A construction of Bayesian networks from databases based on an MDL principle. In *Uncertainty in Artificial*

Intelligence (pp. 266-273). Morgan Kaufmann.

[7] Lam, W., & Bacchus, F. (1994). Learning Bayesian belief networks: An approach based on the MDL principle. *Computational intelligence*, *10*(3), 269-293.

[8] Akaike, H. (1974). A new look at the statistical model identification. In *Selected Papers of Hirotugu Akaike* (pp. 215-222). Springer, New York, NY.

[9] Carvalho, A. M. (2009). Scoring functions for learning Bayesian networks. *Inesc-id Tec. Rep*, *12*.

[10] Cooper, G. F. (1990). The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial intelligence*, *42*(2-3), 393-405.

[11] Chow, C., & Liu, C. (1968). Approximating discrete probability distributions with dependence trees. *IEEE transactions on Information Theory*, *14*(3), 462-467.

[12] Niinimaki, T., & Parviainen, P. (2012). Local structure discovery in Bayesian networks. *arXiv preprint arXiv:1210.4888*.

[13] Gao, T., Fadnis, K., & Campbell, M. (2017, August). Local-to-global Bayesian network structure learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70* (pp. 1193-1202). JMLR. org.

[14] Silander, T., & Myllymaki, P. (2012). A simple approach for finding the globally optimal Bayesian network structure. *arXiv preprint arXiv:1206.6875*.

[15] Chickering, D. M., Heckerman, D., & Meek, C. (2004). Large-sample learning of Bayesian networks is NP-hard. *Journal of Machine Learning Research*, *5*(Oct), 1287-1330.

[16] Tsamardinos, I., Brown, L. E., & Aliferis, C. F. (2006). The max-min hill-climbing Bayesian network structure learning algorithm. *Machine learning*, *65*(1), 31-78.

[17] Ott, S., Imoto, S., & Miyano, S. (2003). Finding optimal models for small gene networks. In *Biocomputing 2004* (pp. 557-567).

[18] Koivisto, M., & Sood, K. (2004). Exact Bayesian structure discovery in Bayesian networks. *Journal of Machine Learning Research*, *5*(May), 549-573.

[19] Tamada, Y., Imoto, S., & Miyano, S. (2011). Parallel algorithm for learning optimal Bayesian network structure. *Journal of Machine Learning Research*, *12*(Jul), 2437-2459.

[20] Ellis, B., & Wong, W. H. (2008). Learning causal Bayesian network structures from experimental data. *Journal of the American Statistical Association*, *103*(482), 778-789.

[21] Wang, Y., Qian, W., Zhang, S., Liang, X., & Yuan, B. (2015). A learning algorithm for Bayesian networks and its efficient implementation on GPUs. *IEEE Transactions on Parallel and Distributed Systems*, *27*(1), 17-30.

[22] Gao, T., & Ji, Q. (2017). Efficient score-based Markov Blanket discovery. *International Journal of Approximate Reasoning*, *80*, 277-293.

[23] Cooper, G. F. (1990). The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial intelligence*, *42*(2-3), 393-405.

[24] Lauritzen, S. L., & Spiegelhalter, D. J. (1988). Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society: Series B (Methodological)*, *50*(2), 157-194.

[25] Zhang, N. L., & Poole, D. (1994, May). A simple approach to Bayesian network computations. In *Proceedings of the Biennial Conference-Canadian Society for Computational Studies of Intelligence* (pp. 171-178). CANADIAN INFORMATION PROCESSING SOCIETY.

[26] Henrion, M. (1988). Propagating uncertainty in Bayesian networks by probabilistic logic sampling. In *Machine Intelligence and Pattern Recognition* (Vol. 5, pp. 149-163). North-Holland.

[27] Yildirim, I. (2012). Bayesian inference: Gibbs sampling. *Technical Note, University of Rochester*.

[28] Liao, Z. A., Sharma, C., Cussens, J., & van Beek, P. (2019, July). Finding all Bayesian network

structures within a factor of optimal. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 33, pp. 7892-7899).

[29] Chen, C., & Yuan, C. (2019). Learning Diverse Bayesian Networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, *33*(01), 7793-7800. https://doi.org/10.1609/aaai.v33i01.33017793

[30] Scutari, M. (2009). Learning Bayesian networks with the bnlearn R package. *arXiv preprint arXiv:0908.3817*.

[31] http://dlib.net/bayes.html

[32] https://github.com/eBay/bayesian-belief-networks

[33] Bouckaert, R. R. (2008). Bayesian network classifiers in weka for version 3-5-7. *Artificial Intelligence Tools*, *11*(3), 369-387.

[34] https://github.com/bmmalone/urlearning-cpp

[35] http://urlearning.org/datasets.html

[36] Campos, C. P. D., & Ji, Q. (2011). Efficient structure learning of Bayesian networks using constraints. *Journal of Machine Learning Research*, *12*(Mar), 663-689.

[37] https://my.eng.utah.edu/~mccully/cs5300lw/

[38] https://artint.info/2e/html/ArtInt2e.Ch8.S6.SS4.html